StoneTor

Everyday Efficiencies

Todd L. Montgomery
@toddlmontgomery

Why should we care?
Understanding (In)Efficiencies
Efficiencies anyone can do

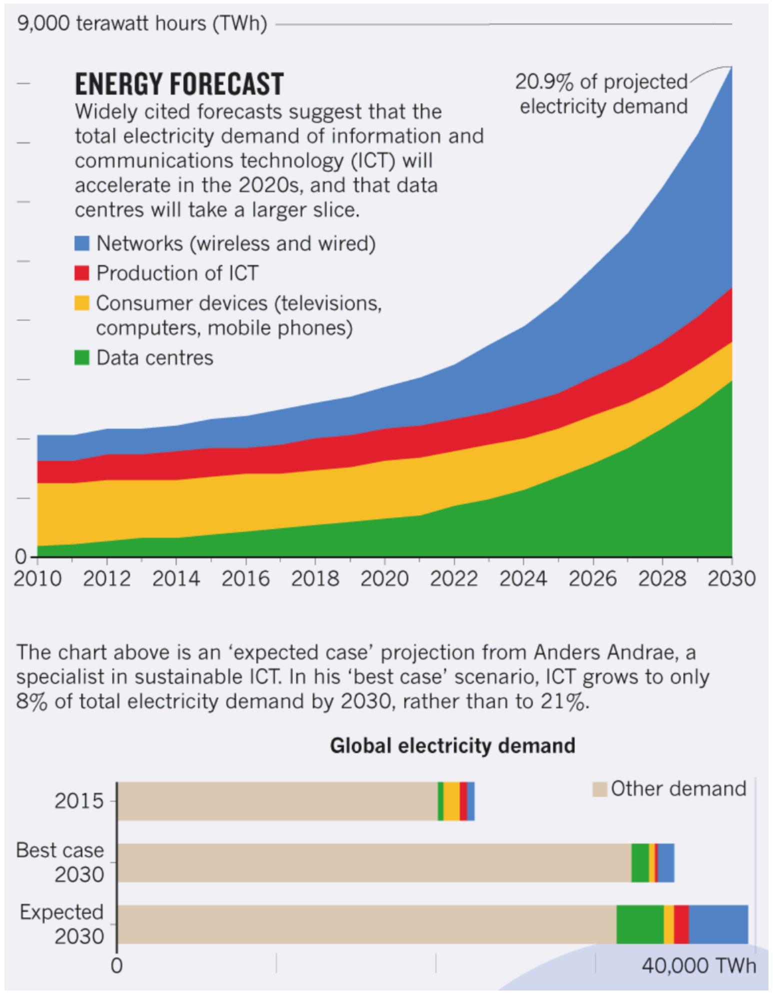
How to stop data centres from gobbling up the world's electricity

Power consumption in data centers is a global problem

20,980 views | Dec 15, 2017, 08:30am

Why Energy Is A Big And Rapidly Growing Problem For Data Centers

Radoslav Danilak Forbes Councils
Forbes Technology Council CommunityVoice



The chart above is an 'expected case' projection from Anders Andrae, a specialist in sustainable ICT. In his 'best case' scenario, ICT grows to only 8% of total electricity demand by 2030, rather than to 21%.

https://www.nature.com/articles/d41586-018-06610-y

https://www.forbes.com/sites/forbestechcouncil/2017/12/15/why-energy-is-a-big-and-rapidly-growing-problem-for-data-centers/#344456665a30

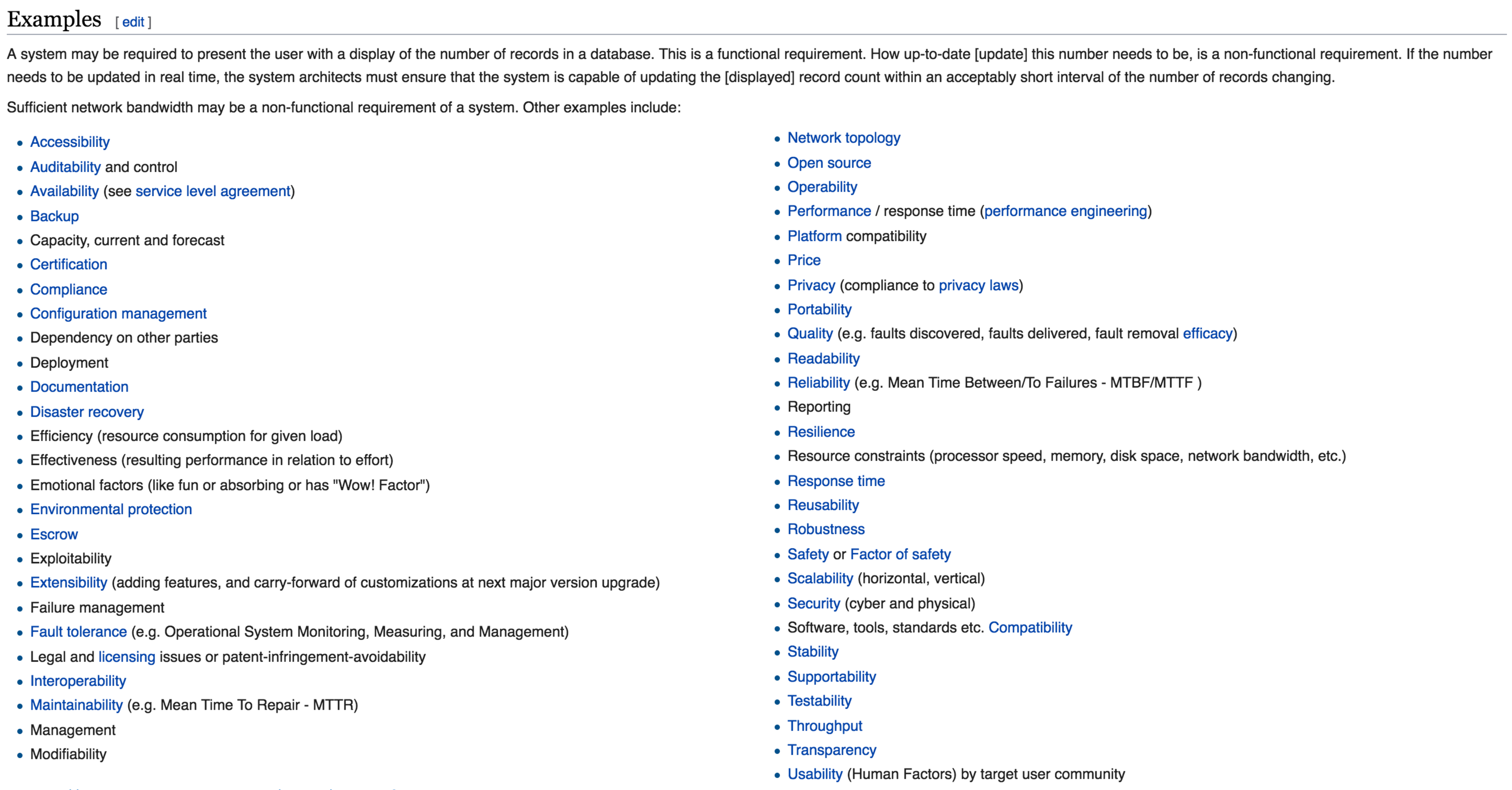
https://www.datacenterdynamics.com/opinions/power-consumption-data-centers-global-problem/

Efficiency/Performance

Non-Functional Requirement

Performance
Quality
Robustness
Safety
Stability
Usability

https://en.wikipedia.org/wiki/Non-functional_requirement



Examples [edit]

A system may be required to present the user with a display of the number of records in a database. This is a functional requirement. How up-to-date [update] this number needs to be, is a non-functional requirement. If the number needs to be updated in real time, the system architects must ensure that the system is capable of updating the [displayed] record count within an acceptably short interval of the number of records changing.

Sufficient network bandwidth may be a non-functional requirement of a system. Other examples include:

- Accessibility
- Auditability and control
- Availability (see service level agreement)
- Backup
- Capacity, current and forecast
- Certification
- Compliance
- Configuration management
- Dependency on other parties
- Deployment
- Documentation
- Disaster recovery
- Efficiency (resource consumption for given load)
- Effectiveness (resulting performance in relation to effort)
- Emotional factors (like fun or absorbing or has "Wow! Factor")
- Environmental protection
- Escrow
- Exploitability
- Extensibility (adding features, and carry-forward of customizations at next major version upgrade)
- Failure management
- Fault tolerance (e.g. Operational System Monitoring, Measuring, and Management)
- Legal and licensing issues or patent-infringement-avoidability
- Interoperability
- Maintainability (e.g. Mean Time To Repair - MTTR)
- Management
- Modifiability
- Network topology
- Open source
- Operability
- Performance / response time (performance engineering)
- Platform compatibility
- Price
- Privacy (compliance to privacy laws)
- Portability
- Quality (e.g. faults discovered, faults delivered, fault removal efficacy)
- Readability
- Reliability (e.g. Mean Time Between/To Failures - MTBF/MTTF)
- Reporting
- Resilience
- Resource constraints (processor speed, memory, disk space, network bandwidth, etc.)
- Response time
- Reusability
- Robustness
- Safety or Factor of safety
- Scalability (horizontal, vertical)
- Security (cyber and physical)
- Software, tools, standards etc. Compatibility
- Stability
- Supportability
- Testability
- Throughput
- Transparency
- Usability (Human Factors) by target user community

https://en.wikipedia.org/wiki/Non-functional_requirement

When not met
is the
system <u>not</u> "Non-Functional"?

“Non”-Functional Requirements Are Unspoken / Incomplete Functional Requirements

Performance (Quality/Security/etc)

At best, an afterthought!

It* isn’t an Issue

…

Until it (suddenly) is

* - Performance/Quality/Security…

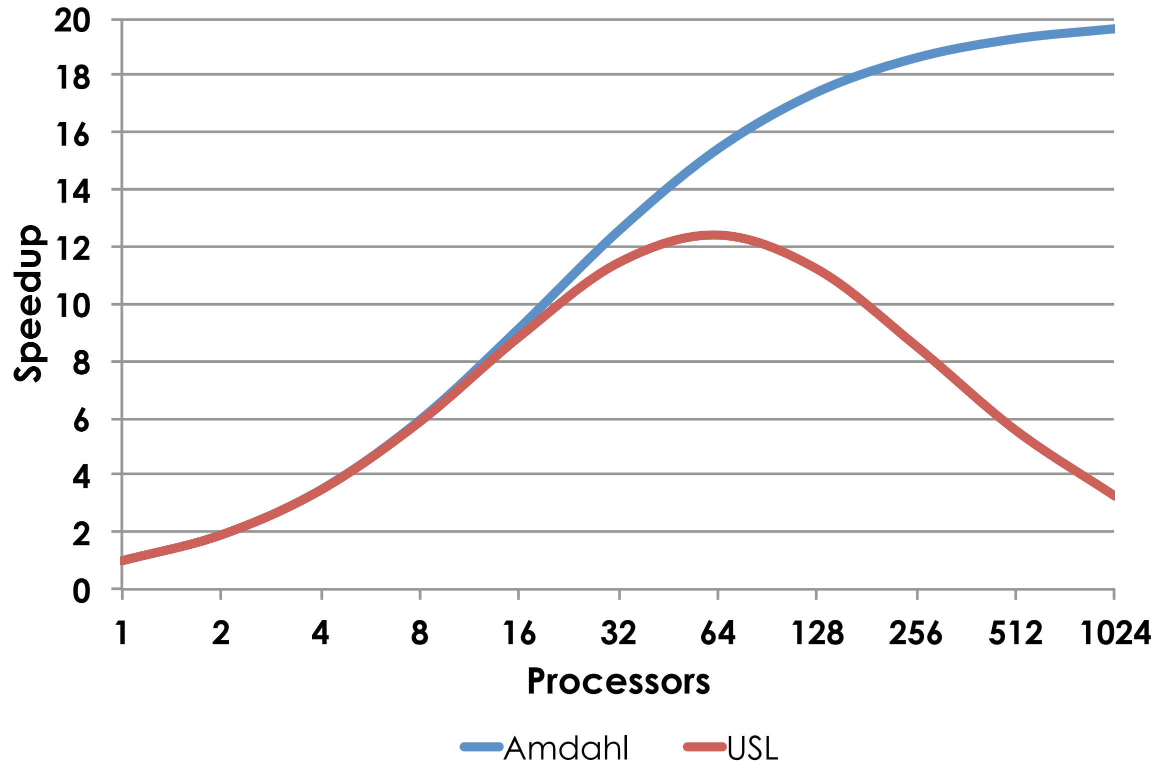
And then…

It is often too late

In the age of cloud…

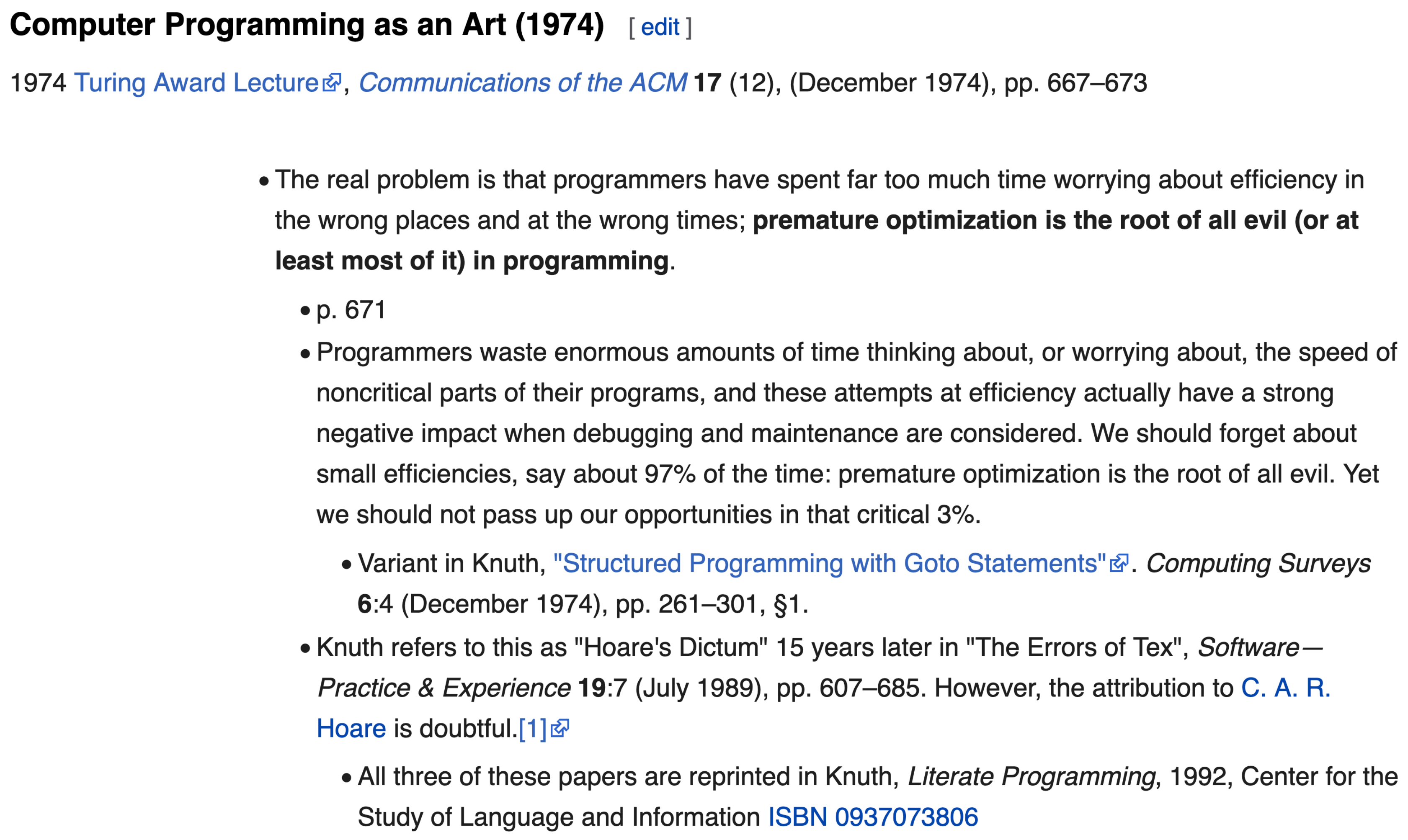
Just throw machines at it

Universal Scalability Law



That <u>Real</u> Quote on

“Premature Optimization” and the root of all evil

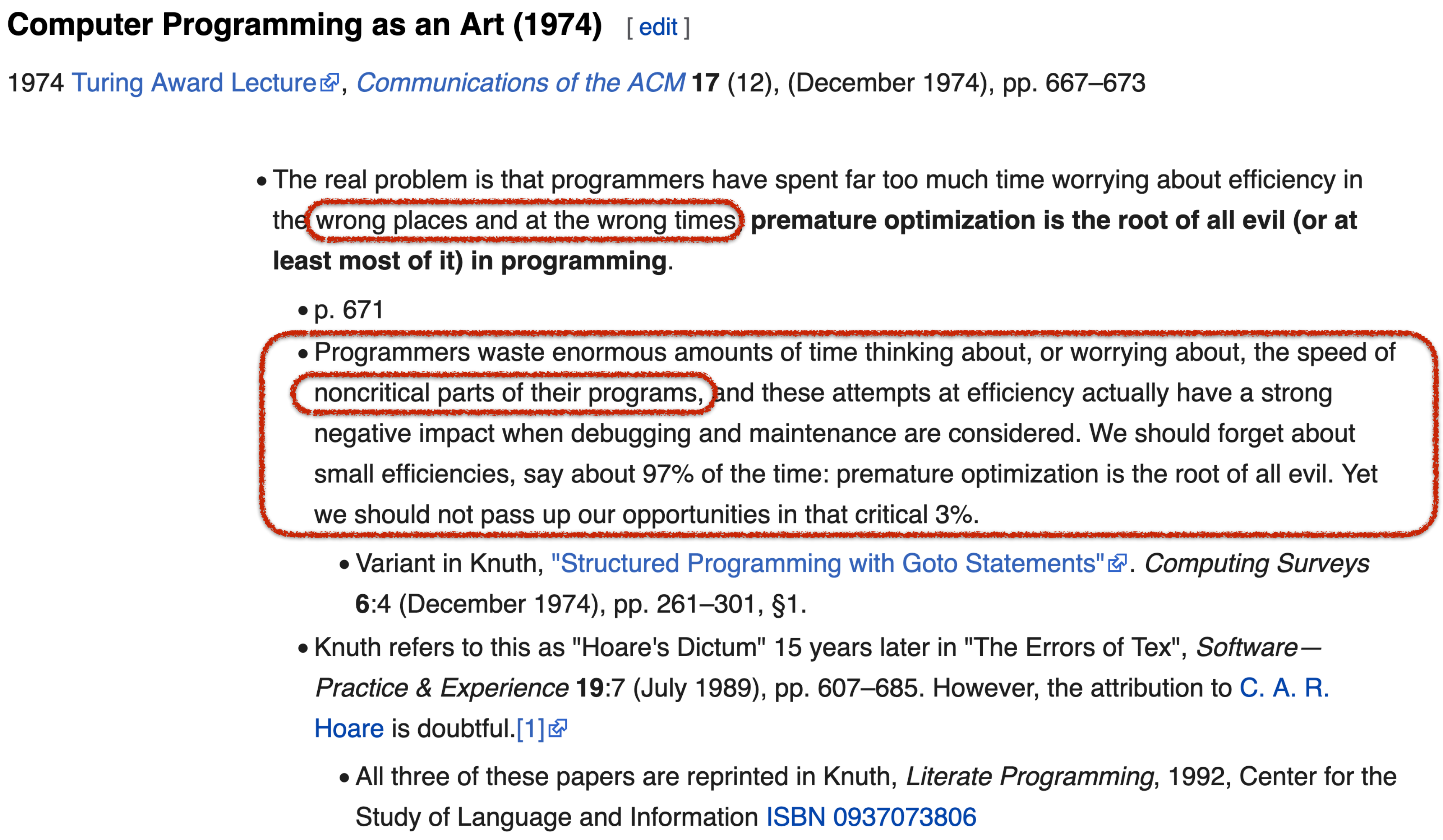


Computer Programming as an Art (1974) [edit]

1974 Turing Award Lecture, *Communications of the ACM* **17** (12), (December 1974), pp. 667–673

- The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; **premature optimization is the root of all evil (or at least most of it) in programming**.
  - p. 671
  - Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.
    - Variant in Knuth, "Structured Programming with Goto Statements". *Computing Surveys* **6**:4 (December 1974), pp. 261–301, §1.
  - Knuth refers to this as "Hoare's Dictum" 15 years later in "The Errors of Tex", *Software—Practice & Experience* **19**:7 (July 1989), pp. 607–685. However, the attribution to C. A. R. Hoare is doubtful.[1]
    - All three of these papers are reprinted in Knuth, *Literate Programming*, 1992, Center for the Study of Language and Information ISBN 0937073806

https://en.wikiquote.org/wiki/Donald_Knuth



Computer Programming as an Art (1974) [edit]

1974 Turing Award Lecture, *Communications of the ACM* **17** (12), (December 1974), pp. 667–673

- The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; **premature optimization is the root of all evil (or at least most of it) in programming**.
  - p. 671
  - Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.
    - Variant in Knuth, "Structured Programming with Goto Statements". *Computing Surveys* **6**:4 (December 1974), pp. 261–301, §1.
  - Knuth refers to this as "Hoare's Dictum" 15 years later in "The Errors of Tex", *Software—Practice & Experience* **19**:7 (July 1989), pp. 607–685. However, the attribution to C. A. R. Hoare is doubtful.[1]
    - All three of these papers are reprinted in Knuth, *Literate Programming*, 1992, Center for the Study of Language and Information ISBN 0937073806

https://en.wikiquote.org/wiki/Donald_Knuth

Pareto Principle

80/20 Rule

https://en.wikipedia.org/wiki/Pareto_principle

Let Data Guide “Where”

"But it doesn't have to be fast!!!"

“But it doesn’t have to be fast!!!”

Doesn’t have to be SLOW either!

“But it doesn’t have to be fast!!!”
“But it doesn’t have to be secure!!!”
“But it doesn’t have to _____!!!”

“But it doesn’t have to <u>WORK</u>!!!??“

We seem to assume speed/security/quality/etc.

is a “special” characteristic added… later

"But it doesn't have to be _____*!!!"

"…It's not <u>my</u> fault!"

* - Fast/Work/Secure…

Other Engineering Disciplines

Top speed of Sedan vs. F1

2x? 3x? 10x?

Do our systems do 100M, 30M, 3K, or 300 tps?

Why are things inefficient?

Not Enough Time?

Too “Lazy”?

Gap(s) in Knowledge?

Too Much Complexity?

End Result

Bad
Design
Choices

Design

Performance
Quality
Security

Start with Design

Everyday Efficiencies

Be Lazy

Don’t reward bad ideas

Don’t be Naive

Good Engineering is Laziness

Too lazy to do something complicated
Never too lazy to stop making it better

Don’t reward bad ideas

Don’t let bad ideas stay around
Don’t be afraid to move on
Don’t be afraid to try something new

Absolutes are for the naive

Always use X!
Never use Y!

Better: Favor X over Y

Concrete Suggestions

Ownership, Dependency, & Coupling

Complexity Kills

Layers of Abstraction are not free

Manage Your Resources

Understand Your Tools
(OS, language, CPU, disk, libs, etc.)

The Compiler is BETTER than you

Idioms Matter

Abstract Later

Design for Composition

Counted vs. Uncounted Loops
Predictable Branches
Simple Conditionals
Stack Allocation
Favor Arrays over Lists
Primitive Data Structures

Everyday Efficiencies

Be Lazy
Don’t reward bad ideas
Don’t be Naive
All starts with Design

Questions?

StoneTor

Twitter: **@toddlmontgomery**

Thank You!